苏州大学实验报告

院、系	计算机学院	年级专业	软件工程	姓名	朱金涛	学号	2327406014
课程名称	机器学习综合实践					成绩	
指导教师	李俊涛	同组实验者	无		实验日期	2025 年 11 月 12 日	

实 验 名 称　聚类与降维算法实践

一. 实验目的

本次实验旨在通过三个具体的应用案例，深入掌握无监督学习的核心技术：

1. **掌握 K-Means 算法**：将其应用于高维数据（手写数字）和三维像素空间（图像色彩），理解其聚类原理和矢量量化（VQ）的应用。
2. **掌握 K-Means 评估**：学习使用多种内、外部评估指标来科学地衡量聚类效果和比较不同初始化策略的优劣。
3. **掌握 PCA 降维**：理解 PCA 如何分离信号与噪声，并熟练运用其 transform 和 inverse_transform 流程实现对图像数据的去噪。

二. 实验内容

本次实验包含三个核心模块，均围绕无监督学习算法在图像和数据分析中的应用展开：

1.K-Means 聚类性能评估： 研究 K-Means 算法在手写数字高维数据集上的聚类效果。本部分内容的重点是**对比不同的初始化策略**（如 k-means++ 与 random），并使用多种内、外部评估指标（如 ARI、AMI、轮廓系数等）来**量化分析**它们在聚类速度和准确性上的差异。

2.K-Means 的矢量量化应用： 探索 K-Means 算法在聚类之外的非传统应用。本部分内容将 K-Means 用于**图像颜色量化**（即减少图片中的颜色总数），通过将图像的 RGB 像素空间视为数据点进行聚类，以实现图像的视觉压缩。

3.PCA 降维与图像去噪： 研究主成分分析 (PCA) 在计算机视觉中的应用。本部分内容的核心是利用 PCA 的**降维与重构**特性，从添加了高斯噪声的数字图像中分离出主要信号成分并丢弃噪声成分，以此探索 PCA **实现图像去噪**的可行性与效果。

三. 实验步骤和结果

（1）基于 K-Means 算法实现图片颜色的量化

1. 实验初始化与数据加载

- 导入库：导入 numpy (用于数值和数组操作)，matplotlib.pyplot (用于图像加载

和显示），time（用于计时），以及 sklearn 库中的 KMeans（聚类算法），shuffle（打乱数据），和 pairwise_distances_argmin（计算最近距离）。

- 设定参数：设定关键超参数 n_colors = 64，即量化后的目标颜色数量（K-Means 的 k 值）。
- 加载图像：使用 matplotlib.pyplot.imread 函数读取指定的本地图像文件（aotizhongxin.jpg）。

```
from time import time

import matplotlib.pyplot as plt
import numpy as np

from sklearn.cluster import KMeans
from sklearn.datasets import load_sample_image
from sklearn.metrics import pairwise_distances_argmin
from sklearn.utils import shuffle

n_colors = 64

# Load the Summer Palace photo
picture = plt.imread("aotizhongxin.jpg")
```

2. 数据预处理

- 规范化：将图像的 uint8 整数像素值（范围 0-255）转换为 float64 浮点数，并除以 255，将其规范化到 [0.0, 1.0] 范围。
- 数据重塑（Reshape）：获取原始图像的尺寸（w, h, d），其中 d 为 3（RGB 通道）。随后，将（w, h, 3）的三维图像数组重塑为（w * h, 3）的二维数组 image_array。在此新格式中，每行代表一个像素，每列代表 R, G, B 三个特征值。

```python
# Convert to floats instead of the default 8 bits integer coding. Dividing by
# 255 is important so that plt.imshow works well on float data (need to
# be in the range [0-1])
picture = np.array(picture, dtype=np.float64) / 255

# Load Image and transform to a 2D numpy array.
w, h, d = original_shape = tuple(picture.shape)
assert d == 3
image_array = np.reshape(picture, (w * h, d))
```

3. K-Means 模型训练

- 数据采样（优化）：为加快模型训练速度，首先使用 shuffle 函数从 image_array（全量像素）中随机抽取 1_000 个像素点，构建一个小型训练样本 image_array_sample。
- 模型拟合：初始化 KMeans 模型，设置 n_clusters=n_colors（64）和 random_state=0（确保结果可复现）。在该小型样本上调用 .fit() 方法进行训练，并记录训练耗时。
- 获取调色板：训练完成后，kmeans.cluster_centers_ 属性即为算法找到的 64 个簇中心（RGB 坐标），它们构成了 K-Means 量化调色板。

```python
print("Fitting model on a small sub-sample of the data")
t0 = time()
image_array_sample = shuffle(image_array, random_state=0, n_samples=1_000)
kmeans = KMeans(n_clusters=n_colors, random_state=0).fit(image_array_sample)
print(f"done in {time() - t0:0.3f}s.")
```

4. K-Means 预测与全图像素替换

- 全量预测：使用上一步训练好的 kmeans 模型，调用 .predict() 方法对完整的 image_array（所有像素）进行预测。此步骤为每个像素分配一个最近的簇索引（0-63）。
- 记录耗时：记录 K-Means 预测全量数据所需的 predict 时间。

```
# Get labels for all points
print("Predicting color indices on the full image (k-means)")
t0 = time()
labels = kmeans.predict(image_array)
print(f"done in {time() - t0:0.3f}s.")
```

5. 基线模型（随机采样）对比

- 构建随机调色板：作为对比基线，使用 shuffle 函数从 image_array 中随机抽取 n_colors (64) 个像素，直接将其作为“随机调色板” (codebook_random)。
- 分配标签：使用 pairwise_distances_argmin 函数，计算 image_array 中的每个像素点到这个“随机调色板”的最近距离，并分配标签。
- 记录耗时：记录随机采样方法分配标签所需的 predict 时间。

```
codebook_random = shuffle(image_array, random_state=0, n_samples=n_colors)
print("Predicting color indices on the full image (random)")
t0 = time()
labels_random = pairwise_distances_argmin(codebook_random, image_array, axis=0)
print(f"done in {time() - t0:0.3f}s.")
```

6. 图像重建

- 定义重建函数：定义 recreate_image 函数。该函数接受调色板（codebook）和标签（labels）作为输入，通过 codebook[labels] 索引操作，将压缩的标签列表“还原”为包含 RGB 值的像素列表。
- 重塑回原状：在 recreate_image 函数内部，将还原的 (w * h, 3) 像素列表重塑回原始图像尺寸 (w, h, 3)。

```
def recreate_image(codebook, labels, w, h):
    """Recreate the (compressed) image from the code book & labels"""
    return codebook[labels].reshape(w, h, -1)
```

7. 结果可视化

- 显示原始图像：使用 plt.imshow() 显示预处理前的原始图像 picture。
- 显示 K-Means 结果：调用 recreate_image(kmeans.cluster_centers_, labels, w, h) 重建 K-Means 量化后的图像，并使用 plt.imshow() 显示。
- 显示随机采样结果：调用 recreate_image(codebook_random, labels_random, w, h) 重建随机采样量化后的图像，并使用 plt.imshow() 显示。
- 对比分析：将三幅图像并排展示，以直观对比 K-Means 算法和随机采样方法在颜色量化效果上的差异。

```
# Display all results, alongside original image
plt.figure(1)
plt.clf()
plt.axis("off")
plt.title("Original image (96,615 colors)")
plt.imshow(picture)

plt.figure(2)
plt.clf()
plt.axis("off")
plt.title(f"Quantized image ({n_colors} colors, K-Means)")
plt.imshow(recreate_image(kmeans.cluster_centers_, labels, w, h))

plt.figure(3)
plt.clf()
plt.axis("off")
plt.title(f"Quantized image ({n_colors} colors, Random)")
plt.imshow(recreate_image(codebook_random, labels_random, w, h))
plt.show()
```

结果展示：

Original image (96,615 colors)

Quantized image (64 colors, K-Means)

（二）基于 K-Means 对手写体数字进行聚类分析

1. 下载手写体数字数据集

```
data, labels = load_digits(return_X_y=True)
(n_samples, n_features), n_digits = data.shape, np.unique(labels).size
```

2. 定义评估基准

```
results += [m(labels, estimator[-1].labels_) for m in clustering_metrics]

# 不需要真实标签的指标（轮廓系数）
results += [
    metrics.silhouette_score(
        data,
        estimator[-1].labels_,
        ...
    )
]
```

接收一个 K-Means 算法实例、一个名称、数据集和真实标签。然后，它建立一个处理管道，该管道首先使用 StandardScaler 对数据进行标准化（使其均值为 0，方差为 1），然后再执行 K-Means 聚类。它会测量拟合数据所需的时间，并计算一系列复杂的聚类评估指标（如 V-measure、调整后兰德指数、轮廓系数等），最后将算法的名称、耗时、惯性和所有评估分数一起打印出来。

3. 运行基准测试

```
# 1. 使用 "k-means++" 策略
kmeans = KMeans(init="k-means++", n_clusters=n_digits, n_init=4, random_state=0)
bench_k_means(kmeans=kmeans, name="k-means++", data=data, labels=labels)

# 2. 使用 "random" 策略
kmeans = KMeans(init="random", n_clusters=n_digits, n_init=4, random_state=0)
bench_k_means(kmeans=kmeans, name="random", data=data, labels=labels)

# 3. 使用 "PCA-based" 策略 (先PCA拟合，再将其主成分作为KMeans的初始中心)
pca = PCA(n_components=n_digits).fit(data)
kmeans = KMeans(init=pca.components_, n_clusters=n_digits, n_init=1)
bench_k_means(kmeans=kmeans, name="PCA-based", data=data, labels=labels)
```

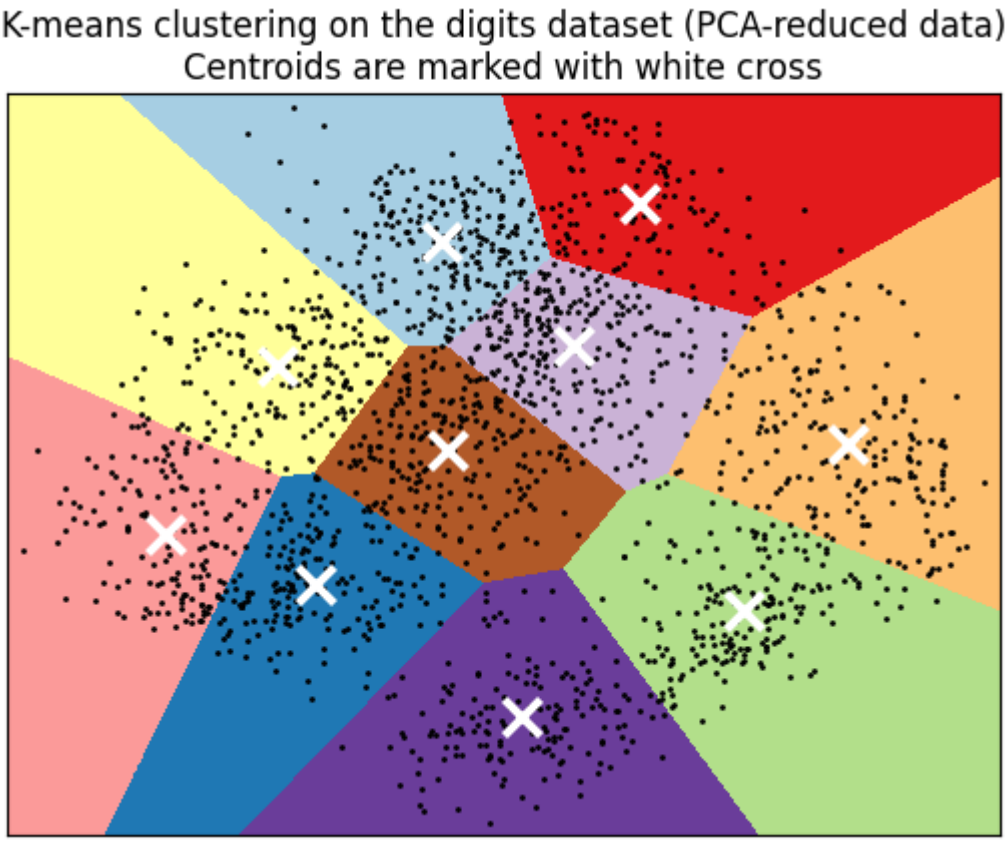
首先打印一个用于显示结果的表格标题行。接着，它分别运行了三种初始化方法：

1. **"k-means++"**：一种标准的、试图优化初始中心点分布的策略。
2. **"random"**：完全随机地选择初始中心点。
3. **"PCA-based"**：先对数据进行 PCA（主成分分析）降维，然后使用 PCA 找到的主成分（pca.components_）作为 K-Means 的初始聚类中心。

结果展示：

init	time	inertia	homo	compl	v-meas	ARI	AMI	silhouette
k-means++	0.190s	69545	0.598	0.645	0.621	0.469	0.617	0.147
random	0.104s	69735	0.681	0.723	0.701	0.574	0.698	0.174
PCA-based	0.031s	69513	0.600	0.647	0.622	0.468	0.618	0.158

4. 将结果可视化为 PCA 降为后的数据



（三）利用 PCA 降维对图像去噪

1. 通过 OpenML 下载图像数据集。

2. 划分训练集、数据集。

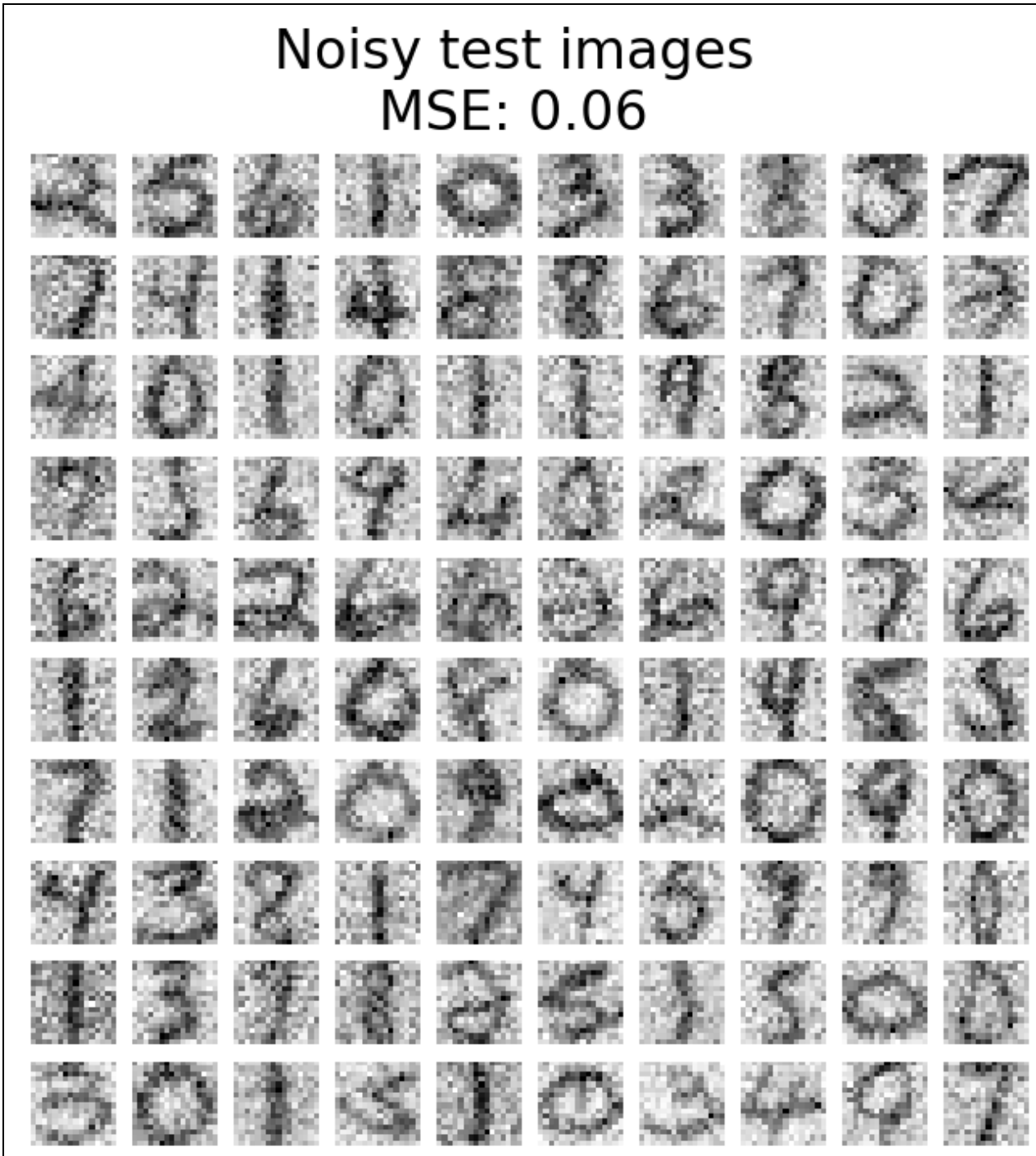
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, stratify=y, random_state=0, train_size=1_000, test_size=100
)
```

训练集 1000，测试集 100

3. 通过绘制测试图像来定性评估图像重建效果

```python
def plot_digits(X, title):
    """Small helper function to plot 100 digits."""
    fig, axs = plt.subplots(nrows=10, ncols=10, figsize=(8, 8))
    for img, ax in zip(X, axs.ravel()):
        ax.imshow(img.reshape((16, 16)), cmap="Greys")
        ax.axis("off")
    fig.suptitle(title, fontsize=24)
```

首先看一下无噪声图像和有噪声图像之间的区别

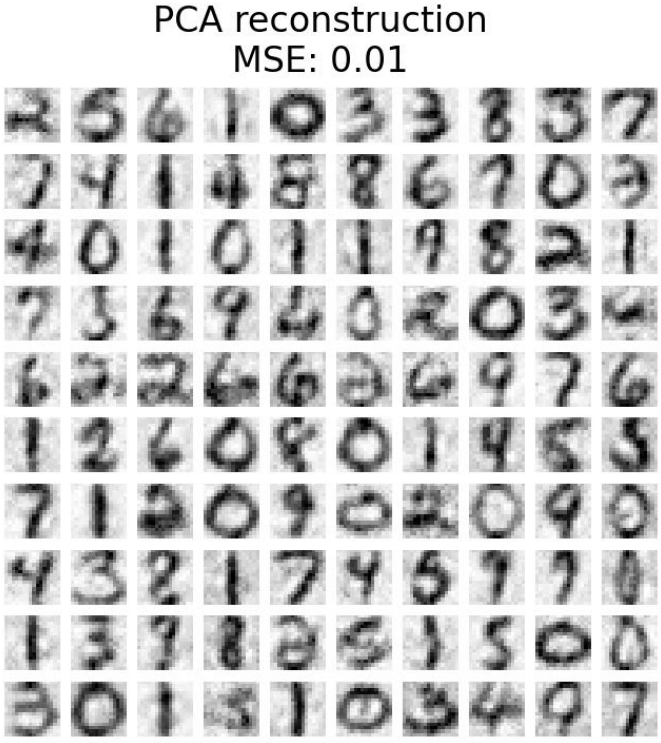


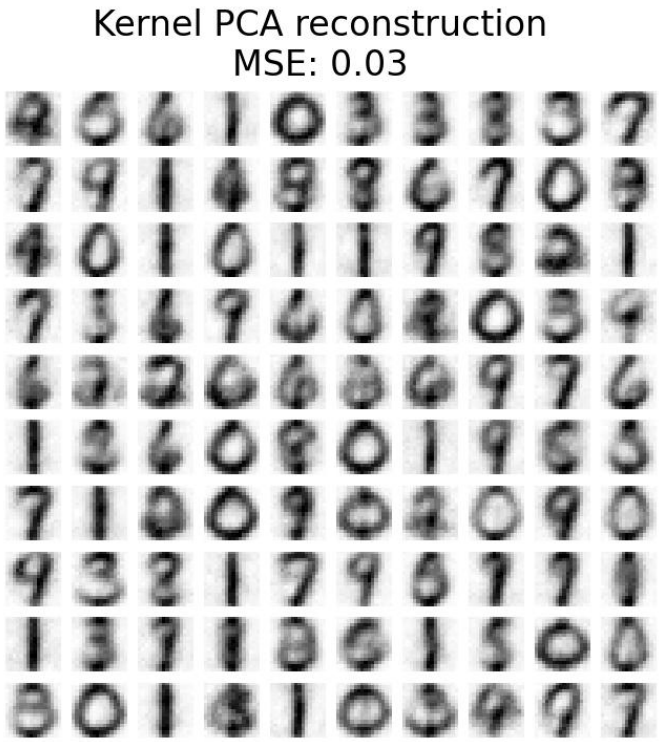
4. 使用线性 PCA 和使用径向基函数（RBF）核的 PCA 核来学习 PCA 基

```python
pca = PCA(n_components=32, random_state=42)
kernel_pca = KernelPCA(
    n_components=400,
    kernel="rbf",
    gamma=1e-3,
    fit_inverse_transform=True,
    alpha=5e-3,
    random_state=42,
)
```

5. 重建和去噪测试图像

结果展示：





根据结果图像也可以看出,使用径向基函数（RBF）核的去噪效果明显优于面向线性 PCA 的。这是因为**线性 PCA**（Linear PCA）假设数据的主要结构是线性的，它只能在数据中找到一个“扁平”的低维超平面来重建数据；如果数

据的真实结构是**非线性的**（例如，手写数字在像素空间中的变化流形是弯曲的），线性 PCA 就无法准确捕捉它，导致在去噪时会错误地将非线性信号的一部分也当成噪声丢弃，或者保留了与信号混杂在一起的噪声。相比之下，**RBF 核 PCA** 通过“核技巧”将数据映射到一个更高维的空间，RBF 核尤其擅长捕捉数据的局部和非线性特征，在这个高维空间中，原本弯曲的信号结构可能被“展开”并与噪声更清晰地分离，因此它能更精确地重构出原始的非线性信号流形，从而达到更优越的去噪效果。

四. 实验总结

本次实验围绕无监督学习的两大核心技术——K-Means 聚类和 PCA 降维，设计并完成了三个核心模块的实践。通过本次实验，我深入掌握了这两种算法的原理、应用及其评估方法。

在 K-Means 聚类方面，本实验从两个维度进行了探索：

1. **矢量量化应用（图像色彩压缩）**：我成功地将 K-Means 算法应用于图像的 RGB 像素空间。通过将数百万个像素点聚类为 64 个簇（n_colors = 64），并用这 64 个簇中心（cluster_centers_）作为新的调色板来重建图像。实验结果直观地显示，K-Means 算法（即便是基于 1000 个点的采样训练 ）所生成的量化图像，在视觉保真度和色彩过渡上，均显著优于随机采样调色板的基线模型。这证明了 K-Means 在矢量量化（VQ）领域的有效性。
2. **聚类性能评估（手写数字识别）**：我使用了一个评估基准函数（bench_k_means），在手写数字数据集上对比了 "k-means++"、

"random" 和 "PCA-based" 三种不同的初始化策略。实验结果数据显示：

- **效率**：PCA-based 初始化速度最快（0.031s），且获得了最低的惯性（Inertia，69513）。
- **质量**：出乎意料的是，"random" 初始化虽然速度中等（0.104s）且惯性较高，但在所有外部评估指标（如 V-measure: 0.701, ARI: 0.574, 轮廓系数: 0.174）上均取得了最佳分数。"k-means++" 和 "PCA-based" 的指标得分非常接近，但略低于 "random"。
- **规范化**：实验流程中包含的 StandardScaler 也让我认识到，在高维数据聚类前进行数据标准化是保证算法性能的重要步骤。

在 PCA 降维与去噪方面，本实验重点对比了线性 PCA 和 RBF 核 PCA 在图像去噪上的应用。实验结果图像明确显示，**RBF 核的去噪效果明显优于线性 PCA**。通过分析，我理解了其根本原因：手写数字在像素空间中的数据结构（流形）是高度**非线性**的。线性 PCA 只能捕捉线性结构，因此在重构时无法区分非线性的信号和噪声；而 RBF 核 PCA 通过核技巧将数据映射到更高维空间，能更有效地分离并重构出原始的非线性信号，从而实现了更出色的去噪效果。

综上所述，本次实验使我不仅掌握了 K-Means 和 PCA 的调用方法，更深入理解了 K-Means 初始化策略对速度和精度的权衡，以及线性与非线性降维方法在处理复杂真实数据（如图像）时的根本差异和适用场景。